


```
FFFFFFFFF 000000 RRRRRRRR IIIIII 000000 EEEEEEEEE LL EEEEEEEEE MM MM
FFFFFFFFF 000000 RRRRRRRR IIIIII 000000 EEEEEEEEE LL EEEEEEEEE MM MM
FF 00 00 RR RR
FF 00 00 RR RR
FF 00 00 RR RR
FF 00 00 RR RR
FFFFFFFF 00 00 RRRRRRRR IIIIII 00 00 EEEEEEEEE LL EEEEEEEEE MM MM
FFFFFFFF 00 00 RRRRRRRR IIIIII 00 00 EEEEEEEEE LL EEEEEEEEE MM MM
FF 00 00 RR RR IIIIII 00 00 EEEEEEEEE LL EEEEEEEEE MM MM
FF 00 00 RR RR IIIIII 00 00 EEEEEEEEE LL EEEEEEEEE MM MM
FF 00 00 RR RR IIIIII 00 00 EEEEEEEEE LL EEEEEEEEE MM MM
FF 00 00 RR RR IIIIII 00 00 EEEEEEEEE LL EEEEEEEEE MM MM
FF 000000 IIIIII SSSSSSSS
FF 000000 IIIIII SSSSSSSS
LL IIIIII SSSSSSSS
LL IIIIII SSSSSSSS
LL II SS
LL II SS
LL II SS
LL II SS
LL II SSSSSS
LL II SSSSSS
LL II SS
LL II SS
LL II SS
LL II SS
LLLLLLLLLL IIIIII SSSSSSSS
LLLLLLLLLL IIIIII SSSSSSSS
```


(2)	74	DECLARATIONS
(3)	131	CALL-BY-VALUE ENTRY POINT DESCRIPTIONS
(4)	175	CALL-BY-REFERENCE ENTRY POINT DESCRIPTIONS
(7)	483	FOR\$IO_X_SE - Transmit single element by descriptor
(8)	529	FOR\$IO_T_DS - Transmit string element by descriptor
(10)	578	FOR\$IO_T_V_DS - Transmit string element then pop off stack
(12)	647	FOR\$IO_X_DA - Transmit entire array by descriptor
(14)	770	FOR\$IO_X_SB - Transmit contiguous implied-DO list
(15)	837	FOR\$IO_X_NL - Transmit non-contiguous implied-DO list
(20)	1090	ERR_HANDLER - Exception handler for errors

```
0000 1 .TITLE FOR$IO_ELEM ; FORTRAN I/O element transmission
0000 2 .IDENT /2-0477 ; File: FORIOELEM.MAR Edit: SBL2047
0000 3
0000 4
0000 5 *****
0000 6
0000 7 *
0000 8 * COPYRIGHT (c) 1978, 1980, 1982, 1984 BY
0000 9 * DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASSACHUSETTS.
0000 10 * ALL RIGHTS RESERVED.
0000 11 *
0000 12 * THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED AND COPIED
0000 13 * ONLY IN ACCORDANCE WITH THE TERMS OF SUCH LICENSE AND WITH THE
0000 14 * INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE OR ANY OTHER
0000 15 * COPIES THEREOF MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY
0000 16 * OTHER PERSON. NO TITLE TO AND OWNERSHIP OF THE SOFTWARE IS HEREBY
0000 17 * TRANSFERRED.
0000 18 *
0000 19 * THE INFORMATION IN THIS SOFTWARE IS SUBJECT TO CHANGE WITHOUT NOTICE
0000 20 * AND SHOULD NOT BE CONSTRUED AS A COMMITMENT BY DIGITAL EQUIPMENT
0000 21 * CORPORATION.
0000 22 *
0000 23 * DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE OR RELIABILITY OF ITS
0000 24 * SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DIGITAL.
0000 25 *
0000 26 *****
0000 27
0000 28
0000 29 ++
0000 30 FACILITY: FORTRAN Support Library - user callable
0000 31
0000 32 ABSTRACT:
0000 33
0000 34 This module implements all of the FORTRAN I/O list element
0000 35 transmits calls made for each of the elements in a
0000 36 READ/WRITE/DECODE/ENCODE (TYPE, ACCEPT, and PRINT)
0000 37 statements at the user program interface level of
0000 38 abstraction (UPI = 1st level). See FOR$IO_BEG
0000 39 and FOR$IO_END modules for I/O statement initialization
0000 40 and termination, respectively.
0000 41
0000 42 ENVIRONMENT: User access mode; mixture of AST level or not
0000 43
0000 44 AUTHOR: Thomas N. Hastings, CREATION DATE: 02-Mar-77
0000 45
0000 46 MODIFIED BY:
0000 47
0000 48 Thomas N. Hastings, 02-Mar-77 : VERSION 01
0000 49 [Previous edit history removed. SBL 30-Sep-1982]
0000 50 2-040 - Use correct offset for imaginary part of DC and GC in
0000 51 FOR$IO_X_DA. JAW 18-Dec-1980
0000 52 2-041 - Add new element transmitters FOR$IO_X_SB, FOR$IO_X_NL and
0000 53 FOR$IO_X_ML to support collapsed implied-DO lists. JAW
0000 54 30-Jan-1981.
0000 55 2-042 - Implement revised interface to FOR$IO_X_SB and FOR$IO_X_NL.
0000 56 (Formatted lists now always use FOR$IO_X_NL.) Improve flow
0000 57 for the noncontiguous unformatted complex case in FOR$IO_X_NL
```



```
0000 58 : and FOR$IO_X_ML. Correct a bug in the handling of complex
0000 59 : elements in FOR$IO_X_NL. JAW 20-Apr-1981
0000 60 : 2-043 - Add entry point FOR$IO_X_SE, for use in transmitting single-
0000 61 : element lists. JAW 06-May-1981
0000 62 : 2-044 - Remove FOR$IO_X_ML. JAW 10-May-1981
0000 63 : 2-045 - Add optimization for unformatted elements to FOR$IO_X_NL (move
0000 64 : element here and don't call UDF level). JAW 13-Jun-1981
0000 65 : 2-046 - Miscellaneous optimizations and cleanup: Shorten CASEL's at
0000 66 : COM_IO_ELEM; merge IO_FC_COM, IO_DC_COM and IO_GC_COM and add
0000 67 : check for unformatted; merge ARRAY_FC, ARRAY_DC and ARRAY_GC;
0000 68 : simplify FOR$IO_X_SB; further optimize FOR$IO_X_NL at NL1A;
0000 69 : simplify FOR$IO_X_SE and move forward in module so branches
0000 70 : will reach; remove spurious ERR_HANDLER; etc. JAW 05-Jul-1981
0000 71 : 2-047 - Change OT$$$ data structure references to FOR$$$. SBL 30-Sep-1982
0000 72 :--
```

```
0000 74 .SBTTL DECLARATIONS
0000 75
0000 76 :
0000 77 : INCLUDE FILES:
0000 78 :
0000 79 : OTSISB.MAR - ISB offset definitions (in S.MLB)
0000 80 : OTSLUB.MAR - LUB offset definitions (in S.MLB)
0000 81 :
0000 82 :
0000 83 : EXTERNAL SYMBOLS:
0000 84 :
0000 85 : .DSABL GBL ; Force declaration of all externals
0000 86 : .EXTRN FOR$$A_CUR_LUB ; Currently active I/O unit
0000 87 : .EXTRN FOR$$A_UDF_PR1 ; User data formatters
0000 88 : ; Contain word PIC dispatch entries.
0000 89 : .EXTRN FOR$$ERR_ENDHND ; Common I/OERR=/END= error handler
0000 90 :
0000 91 : MACROS:
0000 92 :
0000 93 : $SFDEF ; Stack frame offsets defined
0000 94 : $DSCDEF ; Descriptor symbols
0000 95 : $LUBDEF ; LUB definitions
0000 96 : $ISBDEF ; ISB definitions
0000 97 : $FORPAR ; inter-module definitions
0000 98 :
0000 99 : PSECT DECLARATIONS:
0000 100 :
0000 101 : .PSECT _FOR$CODE PIC,SHR,LONG,EXE,NOWRT
0000 102 :
0000 103 : EQUATED SYMBOLS:
0000 104 :
0000 105 :
00000004 0000 106 elem_val = 4 ; offset of item value
00000004 0000 107 elem_adr = 4 ; offset of item address or descriptor addre
00000004 0000 108 sig_args = 4 ; offset of signal args for handler
00000004 0000 109 item_type = 4 ; offset of type in UDF arg list
00000008 0000 110 item_size = 8 ; offset of size in UDF arg list
0000000C 0000 111 item_addr = 12 ; offset of address in UDF arg list
00000010 0000 112 cpx_flag = 16 ; offset of COMPLEX flag in UDF arg list
00000008 0000 113 count = 8 ; Offset of count in SB or NCB
0000000C 0000 114 stride = 12 ; Offset of stride in NCB
00000008 0000 115 depth = 8 ; Offset of depth in MLB
0000000C 0000 116 count_1 = 12 ; Offset of innermost count in MLB
00000010 0000 117 stride_1 = 16 ; Offset of innermost stride in MLB
0000 118 :
0000 119 : The following entry masks are declared here so that they will be
0000 120 : available to FOR$IO_X_SE, which merges them with its own entry mask.
0000 121 :
00000800 0000 122 T_DS_MASK = ^M<R11> ; Entry mask for FOR$IO_T_DS
0000081C 0000 123 X_DA_MASK = ^M<R2, R3, R4, R11> ; Entry mask for FOR$IO_X_DA
00000800 0000 124 X_SB_MASK = ^M<R11> ; Entry mask for FOR$IO_X_SB
0000 125 :
0000 126 :
0000 127 : OWN STORAGE:
0000 128 :
0000 129 :
```



```
0000 131      .SBTTL CALL-BY-VALUE ENTRY POINT DESCRIPTIONS
0000 132
0000 133 :+
0000 134 : The following routine header serves for all of the
0000 135 : call-by-value entry points.
0000 136 :--
0000 137
0000 138 :++
0000 139 : ABSTRACT:
0000 140 :
0000 141 :     Transmit (WRITE) a single data type element from
0000 142 :     the user I/O list to the output buffer by
0000 143 :     calling the appropriate user data formatter
0000 144 :     (UDF) routine for the current I/O statement.
0000 145 :
0000 146 : FORMAL PARAMETERS:
0000 147 :
0000 148 :     ELEM_VAL.rx.v           element by-value
0000 149 :
0000 150 : IMPLICIT INPUTS:
0000 151 :
0000 152 :     FOR$$A_CUR_LUB          Adr. of current logical
0000 153 :                             unit block (LUB). Used to setup
0000 154 :                             ISB base to get current I/O
0000 155 :                             statement type code.
0000 156 :     ISB$B_STTM_TYPE          I/O statement type code - index
0000 157 :                             to dispatch table entry.
0000 158 :     FOR$$AA_UDF_PR1          Array of user data formatters (UDF
0000 159 :                             level of abstraction.)
0000 160 :
0000 161 : IMPLICIT OUTPUTS:
0000 162 :
0000 163 :     NONE
0000 164 :
0000 165 : SIDE EFFECTS:
0000 166 :
0000 167 :     If an error occurs, it is SIGNALed unless an ERR=
0000 168 :     transfer parameter was specified when the I/O statement
0000 169 :     initialization call was made (see module FOR$IO BEG,
0000 170 :     entry points: FOR$(READ,WRITE) {SF,SO,SU,DF,DO,DU,SL} or
0000 171 :     FOR$(DECODE,ENCODE) {MF,MO}), in which case control
0000 172 :     is transferred to the specified address (after stack unwind.)
0000 173 :--
```

```
0000 175      .SBTTL CALL-BY-REFERENCE ENTRY POINT DESCRIPTIONS
0000 176
0000 177 :+
0000 178 : The following routine header serves for all of the
0000 179 : call-by-reference entry points.
0000 180 :-
0000 181
0000 182 :++
0000 183 : ABSTRACT:
0000 184 :
0000 185 :     Transmit (READ or WRITE) a single data type element from
0000 186 :     the user I/O list to the output buffer by
0000 187 :     calling the appropriate user data formatter
0000 188 :     (UDF) routine for the current I/O statement.
0000 189 :
0000 190 : FORMAL PARAMETERS:
0000 191 :
0000 192 :     ELEM_ADR.xx.r           element by-reference
0000 193 :
0000 194 : IMPLICIT INPUTS:
0000 195 :
0000 196 :     FOR$$A_CUR_LUB          Adr. of current logical
0000 197 :                             unit block (LUB). Used to setup
0000 198 :                             ISB base to get current I/O
0000 199 :                             statement type code.
0000 200 :     ISB$B_STTM_TYPE         I/O statement type code - index
0000 201 :                             to dispatch table entry.
0000 202 :     FOR$$AA_UDF_PR1         Array of user data formatters (UDF
0000 203 :                             level of abstraction.)
0000 204 :
0000 205 : IMPLICIT OUTPUTS:
0000 206 :
0000 207 :     NONE
0000 208 :
0000 209 : SIDE EFFECTS:
0000 210 :
0000 211 :     If an error occurs, it is SIGNALed unless an ERR=
0000 212 :     transfer parameter was specified when the I/O statement
0000 213 :     initialization call was made (see module FOR$IO BEG,
0000 214 :     entry points: FOR$(READ,WRITE) {SF,SO,SU,DF,DO,DU,SL} or
0000 215 :     FOR$(DECODE,ENCODE) {MF,MO}), in which case control
0000 216 :     is transferred to the specified address (after stack ???.)
0000 217 :--
0000 218
```



```
04 AC 0800 0000 220 .ENTRY FOR$IO_B_V, ^M<R11> ; BYTE/LOGICAL*1 by-value
05 DF 0002 221 PUSHAL elem_val(AP) ; push address of value
05 11 0005 222 BRB IO_B_COM ; common code for BYTE
0007 223
04 AC 0800 0007 224 .ENTRY FOR$IO_B_R, ^M<R11> ; BYTE/LOGICAL*1 by-reference
05 DD 0009 225 PUSHL elem_addr(AP) ; push address of value
000C 226
000C 227 IO_B_COM:
01 DD 000C 228 PUSHL #1 ; size for BYTE data type
06 DD 000E 229 PUSHL #DSC$K_DTYPE_B ; data-type code for BYTE
22 11 0010 230 BRB COM_IO_ELEM ; common code for all data-types
0012 231
0012 232
0012 233
04 AC 0800 0012 234 .ENTRY FOR$IO_W_V, ^M<R11> ; INTEGER*2 by-value
05 DF 0014 235 PUSHAL elem_val(AP) ; push address of value
05 11 0017 236 BRB IO_W_COM ; common code for INTEGER*2
0019 237
04 AC 0800 0019 238 .ENTRY FOR$IO_W_R, ^M<R11> ; INTEGER*2 by-reference
05 DD 001B 239 PUSHL elem_addr(AP) ; push address of value
001E 240
001E 241 IO_W_COM:
02 DD 001E 242 PUSHL #2 ; size for INTEGER*2 data type
07 DD 0020 243 PUSHL #DSC$K_DTYPE_W ; data-type code for INTEGER*2
10 11 0022 244 BRB COM_IO_ELEM ; common code for all data-types
0024 245
0024 246
0024 247
04 AC 0800 0024 248 .ENTRY FOR$IO_L_V, ^M<R11> ; INTEGER*4 by-value
05 DF 0026 249 PUSHAL elem_val(AP) ; push address of value
05 11 0029 250 BRB IO_L_COM ; common code for INTEGER*4
002B 251
04 AC 0800 002B 252 .ENTRY FOR$IO_L_R, ^M<R11> ; INTEGER*4 by-reference
05 DD 002D 253 PUSHL elem_addr(AP) ; push address of value
0030 254
0030 255 IO_L_COM:
04 DD 0030 256 PUSHL #4 ; size for INTEGER*4 data type
08 DD 0032 257 PUSHL #DSC$K_DTYPE_L ; data-type code for INTEGER*4
0034 258
0034 259 COM_IO_ELEM:
5B 00000000'GF D0 0034 260 MOVL G^FOR$$A_CUR_LUB, R11 ; R11 -> Current Control Block
4D FC AB 09 E1 003B 261 BBC #LUB$V_UNFORMAT, LUB$W_UNIT_ATTR(R11), XCALL1 ; can't optimize if formatted
0040 262
51 50 B0 AB D0 0040 263 MOVL LUB$A_BUF_PTR(R11), R0 ; R0 -> record buffer
04 AE 50 C1 0044 264 ADDL3 R0, 4(TSP), R1 ; R1 = prototype record buffer pointer
B4 AB 51 D1 0049 265 CMPL R1, LUB$A_BUF_END(R11) ; overflows buffer?
3E 1A 004D 266 BGTRU XCALL1 ; branch if yes
3B FF71 CB E9 004F 267 BLBC ISB$B_STTM_TYPE(R11), RU ; dispatch to read/write code
0054 268
0054 269 ;
0054 270 ; write unformatted. Move users data into the record buffer
0054 271 ;
07 01 04 AE CF 0054 272 CASEL 4(SP), #1, #7 ; dispatch on element size
001C' 0059 273 10$: .WORD WBYTE - 10$
0022' 005B 274 .WORD WWORD - 10$
0000' 005D 275 .WORD
0028' 005F 276 .WORD WLONG - 10$
```



```
0000 0061 277 .WORD
0000 0063 278 .WORD
0000 0065 279 .WORD
002E' 0067 280 .WORD WQUAD - 10$
0069 281
51 08 AE D0 0069 282 WOCTA: MOVL 8(SP), R1 ; Get source address
80 81 7D 006D 283 MOVQ (R1)+, (R0)+ ; Move first quadword
80 61 7D 0070 284 MOVQ (R1), (R0)+ ; Move second quadword
51 11 0073 285 BRB COM
80 08 BE 90 0075 286 WBYTE: MOVB @8(SP), (R0)+
4B 11 0079 287 BRB COM
80 08 BE B0 007B 288 WWORD: MOVW @8(SP), (R0)+
45 11 007F 289 BRB COM
80 08 BE D0 0081 290 WLONG: MOVL @8(SP), (R0)+
3F 11 0085 291 BRB COM
80 08 BE 7D 0087 292 WQUAD: MOVQ @8(SP), (R0)+
39 11 008B 293 BRB COM
008D 294
43 11 008D 295 XCALL1: BRB CALL1
008F 296
008F 297
008F 298 ; read unformatted. Move data from record buffer to users element
008F 299
07 01 04 AE CF 008F 300 RU: CASEL 4(SP), #1, #7 ; dispatch on element size
001C' 0094 301 10$: .WORD RBYTE - 10$
0022' 0096 302 .WORD RWORD - 10$
0000 0098 303 .WORD
0028' 009A 304 .WORD RLONG - 10$
0000 009C 305 .WORD
0000 009E 306 .WORD
0000 00A0 307 .WORD
002E' 00A2 308 .WORD RQUAD - 10$
00A4 309
51 08 AE D0 00A4 310 ROCTA: MOVL 8(SP), R1 ; Get result address
81 80 7D 00A8 311 MOVQ (R0)+, (R1)+ ; Move first quadword
61 80 7D 00AB 312 MOVQ (R0)+, (R1) ; Move second quadword
16 11 00AE 313 BRB COM
08 BE 80 90 00B0 314 RBYTE: MOVB (R0)+, @8(SP)
10 11 00B4 315 BRB COM
08 BE 80 B0 00B6 316 RWORD: MOVW (R0)+, @8(SP)
0A 11 00BA 317 BRB COM
08 BE 80 D0 00BC 318 RLONG: MOVL (R0)+, @8(SP)
04 11 00C0 319 BRB COM
08 BE 80 7D 00C2 320 RQUAD: MOVQ (R0)+, @8(SP)
00C6 321 : BRB COM
B0 AB 50 D0 00C6 322 COM: MOVL R0, LUB$A_BUF_PTR(R11) ; store the updated pointer
04 00CA 323 RET
00CB 324
00CB 325 ; come here if checks for the optimization indicate the UDF must be called.
00CB 326
5B 00000000'GF D0 00CB 327 CALLUDF: MOVL G^FOR$A_CUR_LUB, R11 ; R11 = Current Control Block pointer
50 FF71 CB 9A 00D2 328 CALL1: MOVZBL ISB$B_STTM_TYPE(R11), R0
00D7 329 ; R0 = I/O statement type
50 00000000'GF40 D0 00D7 330 MOVL G^FOR$A_UDF_PRI-<ISB$K_FORSTTYLO*4-4>(R0), R0
00DF 331 ; R0 = signed offset relative to beginning
00DF 332 ; of FOR$A_UDF_PRI.
6D 051E'CF DE 00DF 333 MOVAL W^ERR_HANDLER, (FP) ; set up handler
```



```
00000000'GF40 03 FB 00E4 334 CALLS #3, G^FOR$SAA_UDF_PR1[R0] ; call the UDF level routine.
04 04 00EC 335 RET ; and return to the user
00ED 336
0800 00ED 337 .ENTRY FOR$IO_WU V, ^M<R11> ; LOGICAL*2 by-value
04 AC DF 00EF 338 PUSHAL elem_val(AP) ; push address of value
05 11 00F2 339 BRB IO_WU_COM ; common code for LOGICAL*2
00F4 340
0800 00F4 341 .ENTRY FOR$IO_WU R, ^M<R11> ; LOGICAL*2 by-reference
04 AC DD 00F6 342 PUSHL elem_adr(AP) ; push address of value
00F9 343
00F9 344 IO_WU_COM:
02 DD 00F9 345 PUSHL #2 ; size for LOGICAL*2 data type
03 DD 00FB 346 PUSHL #DSC$K_DTYPE_WU ; data-type code for LOGICAL*2
FF34 31 00FD 347 BRW COM_IO_ELEM ; common code for all data-types
0100 348
0100 349
0100 350
0800 0100 351 .ENTRY FOR$IO_LU V, ^M<R11> ; LOGICAL*4 by-value
04 AC DF 0102 352 PUSHAL elem_val(AP) ; push address of value
05 11 0105 353 BRB IO_LU_COM ; common code for LOGICAL*4
0107 354
0800 0107 355 .ENTRY FOR$IO_LU R, ^M<R11> ; LOGICAL*4 by-reference
04 AC DD 0109 356 PUSHL elem_adr(AP) ; push address of value
010C 357
010C 358 IO_LU_COM:
04 DD 010C 359 PUSHL #4 ; size for LOGICAL*4 data type
04 DD 010E 360 PUSHL #DSC$K_DTYPE_LU ; data-type code for LOGICAL*4
FF21 31 0110 361 BRW COM_IO_ELEM ; common code for all data-types
0113 362
0113 363
0113 364
0800 0113 365 .ENTRY FOR$IO_F V, ^M<R11> ; REAL*4 by-value
04 AC DF 0115 366 PUSHAL elem_val(AP) ; push address of value
05 11 0118 367 BRB IO_F_COM ; common code for REAL*4
011A 368
0800 011A 369 .ENTRY FOR$IO_F R, ^M<R11> ; REAL*4 by-reference
04 AC DD 011C 370 PUSHL elem_adr(AP) ; push address of value
011F 371
011F 372 IO_F_COM:
04 DD 011F 373 PUSHL #4 ; size for REAL*4 data type
0A DD 0121 374 PUSHL #DSC$K_DTYPE_F ; data-type code for REAL*4
FF0E 31 0123 375 BRW COM_IO_ELEM ; common code for all data-types
0126 376
0126 377
0126 378
0800 0126 379 .ENTRY FOR$IO_D V, ^M<R11> ; REAL*8 by-value
04 AC DF 0128 380 PUSHAL elem_val(AP) ; push address of value
05 11 012B 381 BRB IO_D_COM ; common code for REAL*8
012D 382
0800 012D 383 .ENTRY FOR$IO_D R, ^M<R11> ; REAL*8 by-reference
04 AC DD 012F 384 PUSHL elem_adr(AP) ; push address of value
0132 385
0132 386 IO_D_COM:
08 DD 0132 387 PUSHL #8 ; size for REAL*8 data type
0B DD 0134 388 PUSHL #DSC$K_DTYPE_D ; data-type code for REAL*8
FEFB 31 0136 389 BRW COM_IO_ELEM ; common code for all data-types
0139 390
```

		0139	391				
04	AC	0800	0139	392	.ENTRY	FOR\$IO_G_V,	^M<R11> ; G REAL*8 by value
		DF	013B	393	PUSHAL	elem_val(AP)	; push address of value
05		11	013E	394	BRB	IO_G_COM	; common code for G REAL*8
			0140	395			
04	AC	0800	0140	396	.ENTRY	FOR\$IO_G_R,	^M<R11> ; G REAL*8 by reference
		DD	0142	397	PUSHL	elem_adr(AP)	; push address of value
			0145	398			
			0145	399		IO_G_COM:	
08		DD	0145	400	PUSHL	#8	; size for G REAL*8 data type
1B		DD	0147	401	PUSHL	#DSC\$K_DTYPE_G	; data-type code for G REAL*8
FEE8		31	0149	402	BRW	COM_IO_ELEM	; common code for all datatypes
			014C	403			
			014C	404			
			014C	405			
04	AC	0800	014C	406	.ENTRY	FOR\$IO_H_V,	^M<R11> ; H REAL*16 by value
		DF	014E	407	PUSHAL	elem_val(AP)	; push address of value
05		11	0151	408	BRB	IO_H_COM	; common code for H REAL*16
			0153	409			
04	AC	0800	0153	410	.ENTRY	FOR\$IO_H_R,	^M<R11> ; H REAL*16 by reference
		DD	0155	411	PUSHL	elem_adr(AP)	; push address of value
			0158	412			
			0158	413		IO_H_COM:	
10		DD	0158	414	PUSHL	#16	; size for H REAL*16 data type
1C		DD	015A	415	PUSHL	#DSC\$K_DTYPE_H	; data-type code for H REAL*16
FED5		31	015C	416	BRW	COM_IO_ELEM	; common code for all datatypes


```
5C 04 AC 06 0804 015F 418 .ENTRY FOR$IO_FC V, ^M<R2,R11>
                                MOVAL elem_val(AP), AP ; get address of value
                                BRB IO_FC_COM
5C 04 AC 06 0804 0161 419
                                0165 420
                                0167 421
                                0167 422 .ENTRY FOR$IO_FC R, ^M<R2,R11>
5C 04 AC 06 0804 0169 423 MOVL elem_adr(AP), AP ; get address of value
                                016D 424
                                016D 425 IO_FC_COM:
50 0A 00 00 016D 426 MOVL #DSC$K_DTYPE_F, R0 ; R0 = type
51 04 00 00 0170 427 MOVL #4, R1 ; R1 = size
27 11 00 00 0173 428 BRB IO_CPLX_COM ; Join common complex code.
                                0175 429
                                0175 430
                                0175 431
5C 04 AC 06 0804 0175 432 .ENTRY FOR$IO_DC V, ^M<R2,R11>
                                MOVAL elem_val(AP), AP ; get address of value
                                BRB IO_DC_COM
5C 04 AC 06 0804 0177 433
                                017B 434
                                017D 435
5C 04 AC 06 0804 017D 436 .ENTRY FOR$IO_DC R, ^M<R2,R11>
                                MOVL elem_adr(AP), AP ; get address of value
                                017F 437
                                0183 438
50 0B 00 00 0183 439 IO_DC_COM:
11 11 00 00 0183 440 MOVL #DSC$K_DTYPE_D, R0 ; R0 = type
                                0186 441
                                0188 442
                                0188 443
                                0188 444
5C 04 AC 06 0804 0188 445 .ENTRY FOR$IO_GC V, ^M<R2,R11>
                                MOVAL elem_val(AP), AP ; get address of value
                                BRB IO_GC_COM
5C 04 AC 06 0804 018A 446
                                018E 447
                                0190 448
5C 04 AC 06 0804 0190 449 .ENTRY FOR$IO_GC R, ^M<R2,R11>
                                MOVL elem_adr(AP), AP ; get address of value
                                0192 450
                                0196 451
50 1B 00 00 0196 452 IO_GC_COM:
51 08 00 00 0196 453 MOVL #DSC$K_DTYPE_G, R0 ; R0 = type
                                0199 454 IO_DC_GC_COM:
                                0199 455 MOVL #8, R1 ; R1 = size
                                019C 456 IO_CPLX_COM:
5B 6D 051E'CF DE 019C 457 MOVL W^ERR_HANDLER, (FP) ; Set up END=/ERR= handler.
00000000'GF DO 01A1 458 MOVL G^FOR$A CUR_LUB, R11 ; R11 -> Current Control Block
52 52 FF71 CB 9A 01A8 459 MOVZBL ISB$B STIM TYPE(R11), R2 ; Get statement type for dispatch.
00000000'GF42 DO 01AD 460 MOVL G^FOR$AA_UDF_PR1-<ISB$K_FORSTTYLO*4-4>[R2], R2
                                01B5 461 ; R2 = displacement to UDF routine
22 FC AB 09 E0 01B5 462 BBS #LUB$V_UNFORMAT, LUB$W_UNIT_ATTR(R11), 20$
                                01BA 463 ; Branch if unformatted.
                                00 DD 01BA 464 PUSHL #0 ; Flag real part of value.
5C DD 01BC 465 PUSHL AP ; Push address of real part.
7E 50 7D 01BE 466 MOVQ R0, -(SP) ; Push size (R1) and type (R0).
04 DD 01C1 467 PUSHL #4 ; Push argument count.
00000000'GF42 6E FA 01C3 468 CALLG (SP), G^FOR$AA_UDF_PR1[R2] ; Transmit real part.
                                01CB 469
                                01CB 470 INCL cpx_flag(SP) ; Flag imaginary part.
0C AE 08 AE CO 01CE 471 ADDL item_size(SP), item_addr(SP) ; Step to imaginary part.
00000000'GF42 6E FA 01D3 472 CALLG (SP), G^FOR$AA_UDF_PR1[R2] ; Transmit imaginary part.
04 01DB 473 RET ; Return to caller.
01DC 474 ;+
```

FOR\$IO_ELEM
2-047

J 6
; FORTRAN I/O element transmission 15-SEP-1984 23:53:43 VAX/VMS Macro V04-00 Page 11
CALL-BY-REFERENCE ENTRY POINT DESCRIPTIO 6-SEP-1984 10:56:44 [FORRTL.SRC]FORIOELEM.MAR;1 (6)

			01DC	475	; Here if unformatted.		
			01DC	476	:-		
	51	5C	DD	01DC	477	20\$:	PUSHL AP
	7E	51	CO	01DE	478		R1, R1
		50	7D	01E1	479		ADDL R1, R1
00000000'GF42		03	FB	01E4	480		MOVQ R0, -(SP)
		04	01EC	481			CALLS #3, G^FOR\$\$AA_UDF_PR1[R2]
							RET

; Push address of item.
; Double the size.
; Push size (R1) and type (R0).
; Call UDF routine.
; Return to caller


```
01ED 483 .SBTTL FOR$IO_X_SE - Transmit single element by descriptor
01ED 484
01ED 485 :++
01ED 486 : ABSTRACT:
01ED 487 :
01ED 488 : Transmit (READ or WRITE) an element which is the only element in
01ED 489 : the current I/O list, without the use of a buffer if possible.
01ED 490 :
01ED 491 : FORMAL PARAMETERS:
01ED 492 :
01ED 493 : DESCR.rr.r Descriptor of class 1, 4, or 191
01ED 494 :
01ED 495 : IMPLICIT INPUTS:
01ED 496 :
01ED 497 : FOR$$A_CUR_LUB Adr. of current logical unit block (LUB)
01ED 498 :
01ED 499 : IMPLICIT OUTPUTS:
01ED 500 :
01ED 501 : ISB$V_SNGL_ELEM Flag indicating that this element is
01ED 502 : the only element in the current I/O list
01ED 503 :
01ED 504 : SIDE EFFECTS:
01ED 505 :
01ED 506 : NONE
01ED 507 :--
01ED 508
01ED 509 .ENTRY FOR$IO_X_SE, T DS_MASK!X_DA_MASK!X_SB_MASK!^M<R11>
5B 00000000'GF 081C 01EF 510 MOVL G^FOR$$A_CUR_LUB, R11 ; RT1 -> Current Control Block
00 96 AB 0C E2 01F6 511 BBSS #ISB$V_SNGL_ELEM, ISB$W_STTM_STAT(R11), 5$
01FB 512 ; Indicate single-element list
01FB 513 ; (potentially unbuffered).
01FB 514 :+
01FB 515 : Dispatch on descriptor class:
01FB 516 : to FOR$IO_X_DA if class = DSC$K_CLASS_A
01FB 517 : to FOR$IO_T_DS if class < DSC$K_CLASS_A (DSC$K_CLASS_S assumed)
01FB 518 : to FOR$IO_X_SB if class > DSC$K_CLASS_A (FOR$K_CLASS_SB assumed)
01FB 519 :-
01FB 520
01FB 521 ASSUME <DSC$K_CLASS_S> LESS_THAN <DSC$K_CLASS_A>
01FB 522 ASSUME <DSC$K_CLASS_A> LESS_THAN <FOR$K_CLASS_SB>
50 04 BC 08 9C 01FB 523 5$: ROTL #8, @4(AP), R0 ; R0<0:7> = descriptor class
04 04 50 91 0200 524 CMPB R0, #DSC$K_CLASS_A ; Is class = 4?
58 13 0203 525 BEQLU FOR$IO_X_DA+2 ; If so, transmit array.
05 1F 0205 526 BLSSU FOR$IO_T_DS+2 ; If less, transmit text string.
0116' 31 0207 527 BRW FOR$IO_X_SB+2 ; Else transmit implied-DO list.
```

```
020A 529 .SBTTL FOR$IO_T_DS - Transmit string element by descriptor
020A 530
020A 531 :++
020A 532 : ABSTRACT:
020A 533 :
020A 534 : Transmit (READ or WRITE) a single character string from
020A 535 : the user I/O list to the output buffer or from
020A 536 : the input buffer to the user I/O list by
020A 537 : calling the appropriate user data formatter
020A 538 : (UDF) routine for the current I/O statement.
020A 539 :
020A 540 : FORMAL PARAMETERS:
020A 541 :
020A 542 : ELEM_ADR.xt.ds element by-descriptor (static)
020A 543 :
020A 544 : IMPLICIT INPUTS:
020A 545 :
020A 546 : FOR$$A_CUR_LUB Adr. of current logical
020A 547 : unit block (LUB). Used to setup
020A 548 : ISB base to get current I/O
020A 549 : statement type code.
020A 550 : ISB$B_STTM_TYPE I/O statement type code - index
020A 551 : to dispatch table entry.
020A 552 : FOR$$A_UDF_PR1 Array of user data formatters (UDF
020A 553 : level of abstraction.)
020A 554 :
020A 555 : IMPLICIT OUTPUTS:
020A 556 :
020A 557 : NONE
020A 558 :
020A 559 : SIDE EFFECTS:
020A 560 :
020A 561 : If an error occurs, it is SIGNALed unless an ERR=
020A 562 : transfer parameter was specified when the I/O statement
020A 563 : initialization call was made (see module FOR$IO_BEG,
020A 564 : entry points: FOR$(READ,WRITE) {SF,SO,SU,DF,DO,DU,SL} or
020A 565 : FOR$(DECODE,ENCODE) {MF,MO}), in which case control
020A 566 : is transferred to the specified address (after stack unwound.)
020A 567 :
020A 568 :
020A 569 :
020A 570 .ENTRY FOR$IO_T_DS, T_DS_MASK
50 04 BC 0800 020C 571 MOVQ @elem_adr(AP), R0 ; get descriptor into R0'R1
7E 51 DD 0210 572 PUSHL R1 ; push address
OE 50 3C 0212 573 MOVZWL R0, -(SP) ; push length
FEB1 0E DD 0215 574 PUSHL #DSC$K_DTYPE_T ; push string data-type
31 0217 575 BRW CALLUDF ; call the UDF
```



```
021A 578 .SBTTL FOR$IO_T_V_DS - Transmit string element then pop off stack
021A 579
021A 580 :++
021A 581 : FUNCTIONAL DESCRIPTION:
021A 582 :
021A 583 : Transmit (READ or WRITE) s single character string from
021A 584 : the user I/O list to the output buffer or from
021A 585 : the input buffer to the user I/O list by
021A 586 : calling the appropriate user data formatter (UDF)
021A 587 : routine for the current I/O statement.
021A 588 : This routine is identical to FOR$IO_T_DS except that
021A 589 : the string passed is popped off the stack as part of the
021A 590 : return to the user program. As such it is a non-standard
021A 591 : procedure. It is really passing the string by value
021A 592 : and is used by the comiler to pass the result of a temporary
021A 593 : string expression computed on the stack.
021A 594 :
021A 595 : CALLING SEQUENCE:
021A 596 :
021A 597 : CALL FOR$IO_T_V_DS (elem_adr.xt.ds)
021A 598 :
021A 599 : INPUT PARAMETERS:
021A 600 : NONE
021A 601 :
021A 602 : IMPLICIT INPUTS:
021A 603 : NONE
021A 604 :
021A 605 : OUTPUT PARAMETERS:
021A 606 : NONE
021A 607 :
021A 608 : IMPLICIT OUTPUTS:
021A 609 : NONE
021A 610 :
021A 611 : COMPLETION CODES:
021A 612 : NONE
021A 613 :
021A 614 : SIDE EFFECTS:
021A 615 :
021A 616 : If an error occurs, it is SIGNALed unless an ERR=
021A 617 : transfer parameter was specified when the I/O statement
021A 618 : initialization call was made (see module FOR$IO_BEG,
021A 619 : entry points: FOR$(DECODE,ENCODE) MF,MO)), in which case control
021A 620 : is transferred to the specified address (after stack unwound).
021A 621 :--
```

```
4800 021A 623 .ENTRY FOR$IO_T_V_DS, ^M<IV, R11>
      021C 624 ; enable integer overflow
      021C 625 MOVAL W^ERR_HANDLER, (FP) ; setup ERR=/END= handler
5B 6D 051E'CF DE 0221 626 MOVL L^FOR$$A_CUR_LUB, R11 ; R11 -> Current Control Block
      50 04 BC 7D 0228 627 MOVQ @elem_adr(AP), R0 ; R0/R1 = string descriptor (len, adr)
      51 DD 022C 628 PUSHL R1 ; arg3 = adr. of string
      7E 50 3C 022E 629 MOVZWL R0, -(SP) ; arg2 = length of string
      OE DD 0231 630 PUSHL #DSC$K_DTYPE_T ; arg1 = data type code of string (text)
50 50 FF71 CB 9A 0233 631 MOVZBL ISB$B_STIM_TYPE(R11), R0 ; get statement type for dispatch
      00000000'GF40 DO 0238 632 MOVL G^FOR$$AA_UDF_PR1-<ISB$K_FORSTTYLO*4-4>[R0], R0 ; R0 = displacement to UDF routine
00000000'GF40 03 FB 0240 633 CALLS #3, G^FOR$$AA_UDF_PR1[R0] ; ; process the element
      50 10 AD DO 0248 635 MOVL SF$L_SAVE_PC(FP), -R0 ; R0 = user PC saved on call
      10 AD 56'AF DE 024C 636 MOVAL B^10$, SF$L_SAVE_PC(FP) ; set return PC to label in this routine
      51 04 BC 3C 0251 637 MOVZWL @elem_adr(AP), R1 ; R1 = length of string to be popped
      04 0255 638 RET ; return to label 10$
      0256 639
      0256 640 ;+
      0256 641 ; Here on RET to pop off string from stack and go back to user
      0256 642 ; -
      0256 643
5E 51 C0 0256 644 10$: ADDL R1, SP ; adjust SP by length of string
      60 17 0259 645 JMP (R0) ; now return to caller
```



```
025B 647 .SBTTL FOR$IO_X_DA - Transmit entire array by descriptor
025B 648
025B 649 :++
025B 650 : ABSTRACT:
025B 651 :
025B 652 : Transmit (READ or WRITE) a single data type element from
025B 653 : the user I/O list to the output buffer by
025B 654 : calling the appropriate user data formatter
025B 655 : (UDF) routine for the current I/O statement.
025B 656 :
025B 657 : FORMAL PARAMETERS:
025B 658 :
025B 659 : ARRAY_DESC_ADR.xx.da Adr. of array descriptor
025B 660 : Data type code in descriptor
025B 661 :
025B 662 : IMPLICIT INPUTS:
025B 663 :
025B 664 : FOR$$A_CUR_LUB Adr. of current logical
025B 665 : unit block (LUB). Used to setup
025B 666 : ISB base to get current I/O
025B 667 : statement type code.
025B 668 : ISB$B_STTM_TYPE I/O statement type code - index
025B 669 : to dispatch table entry.
025B 670 : FOR$$AA_UDF_PR1 Array of user data formatters (UDF
025B 671 : level of abstraction.)
025B 672 :
025B 673 : IMPLICIT OUTPUTS:
025B 674 :
025B 675 : NONE
025B 676 :
025B 677 : SIDE EFFECTS:
025B 678 :
025B 679 : If an error occurs, it is SIGNALed unless an ERR=
025B 680 : transfer parameter was specified when the I/O statement
025B 681 : initialization call was made (see module FOR$IO_BEG,
025B 682 : entry points: FOR$(READ,WRITE) {SF,SO,SU,DF,DO,DU,SL} or
025B 683 : FOR$(DECODE,ENCODE) {MF,MO}), in which case control
025B 684 : is transferred to the specified address (after stack unwound.)
025B 685 :--
```



```
081C 025B 687 .ENTRY FOR$IO X DA, X_DA MASK
DE 025D 688 MOVAL W^ERR_HANDLER, (FP) ; setup ERR=/END= handler
5B 6D 051E^CF DO 0262 689 MOVL G^FOR$SA_CUR_LUB, R11 ; R11 -> Current Control Block
00000000^GF 9A 0269 690 MOVZBL ISB$B_STTM_TYPE(R11), R0 ; get statement type for dispatch
52 50 FF71 CB DO 026E 691 30$: MOVL G^FOR$AA_UDF_PRI-<ISB$K_FORSTTYLO*4-4>(R0), R2
00000000^GF40 DO 0276 692 ; R2 = displacement to UDF routine
50 04 AC DO 0276 693 MOVL elem_adr(AP), R0 ; get ptr to descriptor
54 04 A0 DO 027A 694 MOVL DSC$A_POINTER(R0), R4 ; get base address
09 E0 027E 695 BBS #LUB$V_UNFORMAT, - ; is this unformatted?
45 FC AB 0280 696 LUB$W_UNIT_ATTR(R11), 20$ ; if yes, go transfer the whole array
5C 54 0C A0 C1 0283 697 ADDL3 DSC$L_ARSIZE(R0), R4, AP ; get high address+1
53 60 3C 0288 699 MOVZWL DSC$W_LENGTH(R0), R3 ; get element length
02 A0 0C 91 028B 700 CMPB #DSC$K_DTYPE_FC, DSC$B_DTYPE(R0) ; COMPLEX*8 array?
49 13 028F 701 BEQL ARRAY_FC ; process COMPLEX specially
02 A0 0D 91 0291 702 CMPB #DSC$K_DTYPE_DC, DSC$B_DTYPE(R0) ; D Complex?
4B 13 0295 703 BEQL ARRAY_DC ; special processing
02 A0 1D 91 0297 704 CMPB #DSC$K_DTYPE_GC, DSC$B_DTYPE(R0) ; G Complex?
03 12 029B 705 BNEQ 5$ ; Not complex
0047 31 029D 706 BRW ARRAY_GC ; Special processing
7E 7E D4 02A0 707 5$: CLRL -(SP) ; amake space for elem addr
53 DD 02A2 708 PUSHL R3 ; push element size
02 A0 9A 02A4 709 MOVZBL DSC$B_DTYPE(R0), -(SP) ; push data-type code
02 6E 91 02A8 710 CMPB (SP), #DSC$K_DTYPE_BU ; Did FORTRAN give us BU?
03 12 02AB 711 BNEQ 7$ ; No
6E 06 90 02AD 712 MOVB #DSC$K_DTYPE_B, (SP) ; Yes, it should be type B
03 DD 02B0 713 7$: PUSHL #3 ; 3 arguments to UDF
5C 54 D1 02B2 714 10$: CMPL R4, AP ; element loop point
66 1E 02B5 715 BGEQU ARRAY_RET ; end of array yet?
0C AE 64 DE 02B7 716 MOVAL (R4), item_adr(SP) ; yes
00000000^GF42 6E FA 02BB 717 CALLG (SP), G^FOR$AA_UDF_PRI[R2] ; set element address
54 53 C0 02C3 718 ADDL2 R3, R4 ; call UDF routine
EA 11 02C6 719 BRB 10$ ; add length, point to next element
02C8 720 ; loop back
02C8 721
02C8 722 ;+
02C8 723 ; Here to transmit an entire unformatted array as a single unit
02C8 724 ;+
02C8 725
54 DD 02C8 726 20$: PUSHL R4 ; adr. of first byt of array
0C A0 DD 02CA 727 PUSHL DSC$L_ARSIZE(R0) ; array size in bytes
7E 02 A0 9A 02CD 728 MOVZBL DSC$B_DTYPE(R0), -(SP) ; data type of array elements
00000000^GF42 03 FB 02D1 729 CALLS #3, G^FOR$AA_UDF_PRI[R2] ; call UDF routine
04 02D9 730 RET ; return to user program
02DA 731
02DA 732 ;+
02DA 733 ; Here to handle formatted complex data type. Make two calls per element in array.
02DA 734 ; Indicate which half by fourth actual parameter
02DA 735 ;+
02DA 736
50 0A DO 02DA 737 ARRAY_FC: MOVL #DSC$K_DTYPE_F, R0 ; R0 = type
51 04 DO 02DD 739 MOVL #4, R1 ; R1 = size
0B 11 02E0 740 BRB ARRAY_CPLX_COM ; Join common complex code.
02E2 741
50 0B DO 02E2 742 ARRAY_DC: MOVL #DSC$K_DTYPE_D, R0 ; R0 = type
02E2 743
```



```
03 11 02E5 744 BRB ARRAY_DC_GC_COM ; Join common DC/GC code.
      02E7 745
      02E7 746 ARRAY_GC:
50 1B D0 02E7 747 MOVL #DSC$K_DTYPE_G, R0 ; R0 = type
      02EA 748 ARRAY_DC_GC_COM:
51 0B D0 02EA 749 MOVL #8, R1 ; R1 = size
      02ED 750 ARRAY_CPLX_COM:
      7E 7C 02ED 751 CLRQ -(SP) ; make space for flag and address
7E 50 7D 02EF 752 MOVQ R0, -(SP) ; Push size (R1) and type (R0).
      04 DD 02F2 753 PUSHL #4 ; 4 arguments to UDF routine
      02F4 754
      02F4 755 110$: ; loop
5C 54 D1 02F4 756 CMPL R4, AP ; end of array yet?
      24 1E 02F7 757 BGEQU ARRAY_RET ; yes
      10 AE D4 02F9 758 CLRL cpx_flag(SP) ; flag real part
      0C AE 54 D0 02FC 759 MOVL R4, item_addr(SP) ; push real part address
00000000'GF42 6E FA 0300 760 CALLG (SP), G^FOR$AA_UDF_PR1[R2] ; process real part
      10 AE D6 0308 761 INCL cpx_flag(SP) ; mark imag part
      0C AE 08 AE C0 030B 762 ADDL item_size(SP), item_addr(SP) ; Step to imaginary part.
00000000'GF42 6E FA 0310 763 CALLG (SP), G^FOR$AA_UDF_PR1[R2] ; process imag part
      54 53 C0 0318 764 ADDL2 R3, R4 ; add length
      D7 11 031B 765 BRB 110$ ; loop back
      031D 766
      031D 767 ARRAY_RET:
04 031D 768 RET
```

```
031E 770 .SBTTL FOR$IO_X_SB - Transmit contiguous implied-DO list
031E 771
031E 772 :++
031E 773 : ABSTRACT:
031E 774 :
031E 775 : Transmit (READ or WRITE) the elements of a one-level contiguous
031E 776 : implied-DO list to or from the record buffer, by calling the
031E 777 : appropriate user data formatter (UDF) routine for the current
031E 778 : I/O statement. This entry point is called only for unformatted
031E 779 : statements.
031E 780 :
031E 781 : FORMAL PARAMETERS:
031E 782 :
031E 783 : SIMPLE_BLOCK.rr.r A block describing a simple (contiguous)
031E 784 : implied-DO list, in the form:
031E 785 :
031E 786 : -----
031E 787 : 191 | dtype | length
031E 788 : -----
031E 789 : base address
031E 790 : -----
031E 791 : count
031E 792 : -----
031E 793 :
031E 794 : where count is a signed longword containing the iteration count.
031E 795 : This block is identified by the private-use descriptor class
031E 796 : code 191 = FOR$K_CLASS_SB.
031E 797 :
031E 798 : IMPLICIT INPUTS:
031E 799 :
031E 800 : FOR$$A_CUR_LUB Address of current logical unit block.
031E 801 : ISB$B_STTM_TYPE I/O statement type code - index
031E 802 : to dispatch table entry.
031E 803 : FOR$$AA_UDF_PR1 Array of user data formatters (UDF
031E 804 : level of abstraction.)
031E 805 :
031E 806 : IMPLICIT OUTPUTS:
031E 807 :
031E 808 : NONE
031E 809 :
031E 810 : SIDE EFFECTS:
031E 811 :
031E 812 : Errors are signaled unless an ERR= parameter was specified at
031E 813 : statement initialization time (see FOR$READ_xy, FOR$WRITE_xy),
031E 814 : in which case control is transferred to the specified address,
031E 815 : after stack unwind.
031E 816 : --
031E 817 :
031E 818 : .ENTRY FOR$IO_X_SB, X_SB_MASK
031E 819 : MOVL elem_adr(AP), R12 ; R12 -> argument block
031E 820 : MOVL count(R12), R1 ; R1 = count
031E 821 : BLEQ 10$ ; Return if count <= 0.
031E 822 : +
031E 823 : Construct an argument list on the stack for the call to UDF level.
031E 824 : -
031E 825 : PUSHL DSC$A_POINTER(R12) ; Push element address.
031E 826 : MOVZWL DSC$W_LENGTH(R12), R0 ; Extend element size.
```

5C 04 AC 0800 DO 0320 819
51 08 AC DO 0324 820
2F 15 0328 821
032A 822
032A 823
032A 824
04 AC DD 032A 825
50 6C 3C 032D 826

FOR\$IO_ELEM
2-047

F 7
: FORTRAN I/O element transmission 15-SEP-1984 23:53:43 VAX/VMS Macro V04-00 Page 20
FOR\$IO_X_SB - Transmit contiguous implie 6-SEP-1984 10:56:44 [FORRTL.SRC]FORIOELEM.MAR;1 (14)

7E	50	51	C5	0330	827	MULL3	R1, R0, -(SP)	; Place array size on stack.
	7E	02 AC	9A	0334	828	MOVZBL	DS(\$B_DTYPE(R12), -(SP)	; Push data type code.
	6D	051E'CF	DE	0338	829	MOVAL	W^ERR_HANDLER, (FP)	; Establish ERR=/END= handler.
5B	00000000'GF		D0	033D	830	MOVL	G^FOR\$SA_CUR_LUB, R11	; R11 -> Current Control Block
	51	FF71 CB	9A	0344	831	MOVZBL	ISB\$B_STMT_TYPE(R11), R1	; Get statement type for dispatch
51	00000000'GF41		D0	0349	832	MOVL	G^FOR\$AA_UDF_PRI<<ISB\$K_FORSTTYLO*4-4>[R1], R1	; R1 = displacement to UDF routine
				0351	833			
00000000'GF41		03	FB	0351	834	CALLS	#3, G^FOR\$AA_UDF_PRI[R1]	; Call UDF-level routine.
			04	0359	835 10\$:	RET		; Return to caller.

```
035A 837      .SBTTL FOR$IO_X_NL - Transmit non-contiguous implied-DO list
035A 838
035A 839 :++
035A 840 : ABSTRACT:
035A 841 :
035A 842 : Transmit (READ or WRITE) the elements of a one-level
035A 843 : non-contiguous implied-DO list to or from the record buffer, by
035A 844 : calling the appropriate user data formatter (UDF) routine for
035A 845 : the current I/O statement. This entry point is called for both
035A 846 : formatted and unformatted statements.
035A 847 :
035A 848 : FORMAL PARAMETERS:
035A 849 :
035A 850 : NON_CTG_BLOCK.rr.r      A block describing a non-contiguous
035A 851 :                          implied-DO list, in the form:
035A 852 :
035A 853 :
035A 854 :
035A 855 :
035A 856 :
035A 857 :
035A 858 :
035A 859 :
035A 860 :
035A 861 :
035A 862 :
035A 863 :
035A 864 :
035A 865 :
035A 866 :
035A 867 :
035A 868 :
035A 869 :
035A 870 :
035A 871 :
035A 872 :
035A 873 :
035A 874 :
035A 875 :
035A 876 :
035A 877 :
035A 878 :
035A 879 :
035A 880 :
035A 881 :
035A 882 :
035A 883 :
035A 884 :
035A 885 :
035A 886 :
035A 887 :
035A 888 :
```

190	dtype	length
base address		
count		
stride		

```

035A 863 : where count is a signed longword containing the iteration count,
035A 864 : and stride is a signed longword containing the amount by which
035A 865 : to augment the base address for each element transmitted. This
035A 866 : block is identified by the private-use descriptor class code
035A 867 : 190 = FOR$K_CLASS_NL.
035A 868 :
035A 869 : IMPLICIT INPUTS:
035A 870 :
035A 871 : FOR$$A_CUR_LUB      Address of current logical unit block.
035A 872 : ISB$B_STTM_TYPE     I/O statement type code - index
035A 873 :                      to dispatch table entry.
035A 874 : FOR$$AA_UDF_PR1     Array of user data formatters (UDF
035A 875 :                      level of abstraction.)
035A 876 :
035A 877 : IMPLICIT OUTPUTS:
035A 878 :
035A 879 : NONE
035A 880 :
035A 881 : SIDE EFFECTS:
035A 882 :
035A 883 : Errors are signaled unless an ERR= parameter was specified at
035A 884 : statement initialization time (see FOR$READ_xy, FOR$WRITE_xy),
035A 885 : in which case control is transferred to the specified address,
035A 886 : after stack unwind.
035A 887 : --
035A 888 :
```



```

      5C 04 AC 081C 035A 890      .ENTRY FOR$IO_X_NL, ^M<R2, R3, R4, R11>
      54 0C AC DO 035C 891      MOVL elem_adr(AP), R12      ; R12 -> argument block
      53 08 AC DO 0360 892      MOVL stride(R12), R4      ; R4 = stride
      01 14 0364 893      MOVL count(R12), R3      ; R3 = count
      04 0368 894      BGTR 10$      ; Is count > 0?
      036A 895      RET      ; If not, return.
      036B 896
      6D 051E'CF DE 036B 897 10$: MOVAL W^ERR_HANDLER, (FP)      ; Establish ERR=/END= handler.
      5B 00000000'GF DO 0370 898      MOVL G^FOR$$A_CUR_LUB, R11      ; R11 -> Current Control Block
      52 FF71 CB 9A 0377 899      MOVZBL ISB$B_STTM_TYPE(R11), R2      ; Get statement type for dispatch
      52 00000000'GF42 DO 037C 900      MOVL G^FOR$$AA_UDF_PR1-<ISB$K_FORSTTYLO*4-4>[R2], R2      ; R2 = displacement to UDF routine
      0384 901
      0384 902 ;+
      0384 903 ; Construct an argument list on the stack for the call to UDF level.
      0384 904 ; Allow room for a second argument list, for use if the elements are
      0384 905 ; formatted and complex. Each argument list is composed of five
      0384 906 ; longwords, including the count. (Be careful about pushing anything
      0384 907 ; else after this point.)
      0384 908 ;+
      5E 18 C2 0384 909      SUBL #24, SP      ; Allow room for flag in first
      04 AC DD 0387 910      PUSHL DSC$A_POINTER(R12)      ; list (4), plus second list (20).
      7E 04 AC DD 0387 911      MOVZWL DSC$W_LENGTH(R12), -(SP)      ; Push element address.
      7E 02 AC 9A 038A 912      MOVZBL DSC$B_DTYPE(R12), -(SP)      ; Push element size.
      03  DD 0391 913      MOVZBL DSC$B_DTYPE(R12), -(SP)      ; Push data type code.
      0393 914      PUSHL #3      ; Push argument count.
      0393 915 ;+
      0393 916 ; Determine whether the array can be transmitted as a unit.
      0393 917 ;+
      15 FC AB 09 E1 0393 918      BBC #LUB$V_UNFORMAT, -      ; If formatted, transmit the
      54 6C 10 00 ED 0398 919      LUB$W_UNIT_ATTR(R11), NLFMT      ; elements individually.
      6C 12 0398 920      CMPZV #0, #T6, DSC$W_LENGTH(R12), R4      ; Is length = stride?
      039D 921      BNEQ NL1PER      ; If not, it's noncontiguous.
      039F 922 ;+
      039F 923 ; Unformatted and contiguous: transmit the array with a single call.
      039F 924 ;+
      08 AE 08 AC C4 039F 925 NLUNIT: MULL2 count(R12), item_size(SP)      ; Compute array size.
      00000000'GF42 6E FA 03A4 926      CALLG (SP), G^FOR$$AA_UDF_PR1[R2]      ; Call UDF-level routine.
      04 03AC 927      RET      ; Return to caller.
      03AD 928 ;+
      03AD 929 ; Formatted: check for complex.
      03AD 930 ;+
      50 000C0004 8F 02 AC 78 03AD 931 NLFMT: ASHL DSC$B_DTYPE(R12), -      ; Is this F, D or G complex?
      03B6 932      #<<1 @ <31-DSC$K_DTYPE_FC>> + -      ; [This instruction
      03B6 933      <1 @ <31-DSC$K_DTYPE_DC>> + -      ; shifts a mask having
      03B6 934      <1 @ <31-DSC$K_DTYPE_GC>>], R0      ; bits 19, 18 and 2 set
      03B6 935      ; (representing FC, DC and GC
      03B6 936      ; respectively) "dtype" places.]
      03B6 937      BLSS NL2PER      ; Branch if any of the above.
      02 04 AE 91 03B8 938      CMPB item_type(SP), #DSC$K_DTYPE_BU      ; Is it type BU?
      04 AE 06 90 03BC 939      BNEQ 20$      ; Branch if not.
      0149 31 03BE 940      MOVB #DSC$K_DTYPE_B, item_type(SP)      ; Yes: make it type B.
      03C2 941 20$: BRW NL1B      ; Use NL1PER, but bypass
      03C5 942      ; the optimization applying to
      03C5 943      ; unformatted elements only.
```

```
03C5 945 ;+
03C5 946 ; Formatted and complex. Transmit the elements individually, making two
03C5 947 ; calls per element. Use the flag argument to identify the real or
03C5 948 ; imaginary part.
03C5 949 ;:-
03C5 950 ASSUME <DSC$K_DTYPE_FC - DSC$K_DTYPE_F> EQUAL <DSC$K_DTYPE_DC - DSC$K_DTYPE_
03C5 951 ASSUME <DSC$K_DTYPE_FC - DSC$K_DTYPE_F> EQUAL <DSC$K_DTYPE_GC - DSC$K_DTYPE_
03C5 952 NL2PER: INCL (SP) ; Increase argument count to 4.
03C7 953 SUBL #<DSC$K_DTYPE_FC - DSC$K_DTYPE_F>, -
03CB 954 item_type(SP) ; Convert FC/DC/GC to F/D/G.
08 AE 08 AE FF 8F 78 03CB 955 ASHL #-1, item_size(SP), item_size(SP) ; Halve item size.
10 AE 10 AE 10 AE D4 03D2 956 CLRL cpx_flag(SP) ; Indicate real part.
14 AE 14 AE 6E 7D 03D5 957 MOVQ (SP), 20(SP) ; Initialize second arg list.
1C AE 08 AE D0 03D9 958 MOVL item_size(SP), item_size+20(SP) ; ...
24 AE 01 D0 03DE 959 MOVL #1, cpx_flag+20(SP) ; Indicate imaginary part.
0C AE 04 AC D0 03E2 960 MOVL DSC$A_POINTER(R12), item_addr(SP)
03E7 961 ; Initialize real part address.
20 AE 08 AE 04 AC C1 03E7 962 ADDL3 DSC$A_POINTER(R12), item_size(SP), item_addr+20(SP)
03EE 963 ; Initialize imag part address.
00000000'GF42 6E FA 03EE 964 40$: CALLG (SP), G^FOR$AA_UDF_PR1[R2] ; Transmit real part.
00000000'GF42 14 AE FA 03F6 965 CALLG 20(SP), G^FOR$AA_UDF_PR1[R2] ; Transmit imaginary part.
0C AE 54 C0 03FF 966 ADDL R4, item_addr(SP) ; Stride to next element.
20 AE 54 C0 0403 967 ADDL R4, item_addr+20(SP) ; ...
E4 53 F5 0407 968 SOBGTR R3, 40$ ; Decrement and test count.
04 040A 969 RET ; Return to caller.
040B 970 ;+
040B 971 ; Unformatted and noncontiguous, or formatted and not complex. Transmit
040B 972 ; the elements individually, making one call per element. (Formatted
040B 973 ; enters at NL1B, below.)
040B 974 ;:-
0E 04 AE D1 040B 975 NL1PER: CMPL item_type(SP), #DSC$K_DTYPE_T ; Is this type character?
03 12 040F 976 BNEQ NL1A ; If not, optimize.
00FA 31 0411 977 BRW NL1B ; If so, don't optimize, since
0414 978 ; its length is not restricted
0414 979 ; to 1, 2, 4, 8 or 16 bytes.
0414 980 ;+
0414 981 ; Optimization for unformatted only.
0414 982 ;
0414 983 ; Set R2 to the address of the appropriate MOVx instruction, below, for
0414 984 ; use when transmitting elements which do not require a call to UDF
0414 985 ; level (most of them).
0414 986 ;:-
07 38 FF71 CB E9 0414 987 NL1A: BLBC ISB$B_STTM_TYPE(R11), RADDR ; Branch if reading.
01 08 AE CF 0419 988 CASEL item_size(SP), #1, #7 ; dispatch on element size
0017' 041E 989 10$: .WORD AWBYTE - 10$
001E' 0420 990 .WORD AWORD - 10$
0000 0422 991 .WORD
0025' 0424 992 .WORD AWLONG - 10$
0000 0426 993 .WORD
0000 0428 994 .WORD
0000 042A 995 .WORD
002C' 042C 996 .WORD AWQUAD - 10$
52 04D1'CF DE 042E 997 AWOCTA: MOVAL W^WUOCTA, R2 ; R2 = addr of 'MOV0' for write
52 04B9'CF DE 0433 998 BRB ACOM ; R2 = addr of MOV0 for write
4B 11 0435 999 AWBYTE: MOVAL W^WUBYTE, R2 ; R2 = addr of MOV0 for write
52 04BF'CF DE 043A 1000 BRB ACOM
1001 AWWORD: MOVAL W^WUWORD, R2 ; R2 = addr of MOV0 for write
```



```
52 04C5'CF 44 11 0441 1002 BRB ACOM
DE 0443 1003 AWLONG: MOVAL W^WULONG, R2 ; R2 = addr of MOVL for write
11 0448 1004 BRB ACOM
52 04CB'CF 3D 11 044A 1005 AWQUAD: MOVAL W^WUQUAD, R2 ; R2 = addr of MOVQ for write
36 11 044F 1006 BRB ACOM
07 01 08 AE CF 0451 1007 RADDR: CASEL item_size(SP), #1, #7 ; dispatch on element size
0017' 0456 1009 10$: .WORD ARBYTE - 10$
001E' 0458 1010 .WORD ARWORD - 10$
0000 045A 1011 .WORD
0025' 045C 1012 .WORD ARLONG - 10$
0000 045E 1013 .WORD
0000 0460 1014 .WORD
0000 0462 1015 .WORD
002C' 0464 1016 .WORD ARQUAD - 10$
52 04F5'CF DE 0466 1017 AROCTA: MOVAL W^RUOCTA, R2 ; R2 = addr of 'MOVQ' for read
1A 11 046B 1018 BRB ACOM
52 04DD'CF DE 046D 1019 ARBYTE: MOVAL W^RUBYTE, R2 ; R2 = addr of MOVB for read
13 11 0472 1020 BRB ACOM
52 04E3'CF DE 0474 1021 ARWORD: MOVAL W^RUWORD, R2 ; R2 = addr of MOVW for read
0C 11 0479 1022 BRB ACOM
52 04E9'CF DE 047B 1023 ARLONG: MOVAL W^RULONG, R2 ; R2 = addr of MOVL for read
05 11 0480 1024 BRB ACOM
52 04EF'CF DE 0482 1025 ARQUAD: MOVAL W^RUQUAD, R2 ; R2 = addr of MOVQ for read
0487 1026 BRB ACOM
0487 1027 ACOM:
0487 1028 ;+
0487 1029 ; Here after a call to UDF level to re-establish the pointer in R0.
0487 1030 ;+
50 B0 AB D0 0487 1031 NL1AX: MOVL LUB$A_BUF_PTR(R11), R0 ; R0 -> next buffer location
048B 1032 ;+
048B 1033 ; Here after a simple move, to see whether another move is possible.
048B 1034 ;+
51 08 AE 50 C1 048B 1035 NL1AY: ADDL3 R0, item_size(SP), R1 ; R1 = final byte needed + 1
B4 AB 51 D1 0490 1036 CMPL R1, LUB$A_BUF_END(R11) ; Will element fit in buffer?
02 1A 0494 1037 BGTRU NL1CAL ; Branch if not.
62 17 0496 1038 JMP (R2) ; Go move the element directly.
0498 1039 ;+
0498 1040 ; Here if there is no room for a particular element.
0498 1041 ;+
B0 AB 50 D0 0498 1042 NL1CAL: MOVL R0, LUB$A_BUF_PTR(R11) ; Save possibly updated pointer.
51 FF71 CB 9A 049C 1043 MOVZBL ISB$B_STTM_TYPE(R11), R1 ; Reconstruct dispatch address.
00000000'GF41 D0 04A1 1044 MOVL G^FOR$AA UDF_PR1-<ISB$K_FORSTTYLO*4-4>LR1], R1
00000000'GF41 6E FA 04A9 1045 CALLG (SP), G^FOR$AA UDF_PR1[R1] ; Call UDF routine w/ CALLG.
OC AE 54 C0 04B1 1046 ADDL R4, item_addr(SP) ; Step to next element.
CF 53 F5 04B5 1047 SOBGTR R3, NL1AX ; Decrement and test count.
04 04B8 1048 RET ; Return to caller.
```

```
80 0C BE 90 04B9 1050 WUBYTE: MOVB @item_addr(SP), (R0)+ ; Move byte to buffer
      40 11 04BD 1051 BRB UCOM
80 0C BE 80 04BF 1052 WUWORD: MOVW @item_addr(SP), (R0)+ ; Move word to buffer
      3A 11 04C3 1053 BRB UCOM
80 0C BE D0 04C5 1054 WULONG: MOVL @item_addr(SP), (R0)+ ; Move longword to buffer
      34 11 04C9 1055 BRB UCOM
80 0C BE 7D 04CB 1056 WUQUAD: MOVQ @item_addr(SP), (R0)+ ; Move quadword to buffer
      2E 11 04CF 1057 BRB UCOM
51 0C AE D0 04D1 1058 WUOCTA: MOVL item_addr(SP), R1 ; Move octaword to buffer
      80 81 7D 04D5 1059 MOVQ (R1)+, (R0)+ ; Move first quadword
      80 61 7D 04D8 1060 MOVQ (R1), (R0)+ ; Move second quadword
      22 11 04DB 1061 BRB UCOM
      04DD 1062
0C BE 80 90 04DD 1063 RUBYTE: MOVB (R0)+, @item_addr(SP) ; Move byte from buffer
      1C 11 04E1 1064 BRB UCOM
0C BE 80 80 04E3 1065 RUWORD: MOVW (R0)+, @item_addr(SP) ; Move word from buffer
      16 11 04E7 1066 BRB UCOM
0C BE 80 D0 04E9 1067 RULONG: MOVL (R0)+, @item_addr(SP) ; Move longword from buffer
      10 11 04ED 1068 BRB UCOM
0C BE 80 7D 04EF 1069 RUQUAD: MOVQ (R0)+, @item_addr(SP) ; Move quadword from buffer
      0A 11 04F3 1070 BRB UCOM
51 0C AE D0 04F5 1071 RUOCTA: MOVL item_addr(SP), R1 ; Move octaword from buffer
      81 80 7D 04F9 1072 MOVQ (R0)+, (R1)+ ; Move first quadword
      61 80 7D 04FC 1073 MOVQ (R0)+, (R1) ; Move second quadword
      04FF 1074 BRB UCOM
0C AE 54 C0 04FF 1075 UCOM: ADDL R4, item_addr(SP) ; Step to next element.
      05 53 F5 0503 1076 SOBGTR R3, BNL1AY ; Decrement and test count.
B0 AB 50 D0 0506 1077 MOVL R0, LUB$A_BUF_PTR(R11) ; Update buffer pointer.
      04 050A 1078 RET ; Return to caller.
      FF7D 31 050B 1079 BNL1AY: BRW NL1AY
      050E 1080
      050E 1081 ;+
      050E 1082 ; Formatted and not complex
      050E 1083 ;+
00000000*GF42 6E FA 050E 1084 NL1B: CALLG (SP), G*FOR$AA UDF_PR1[R2] ; Call UDF routine w/ CALLG.
      OC AE 54 C0 0516 1085 ADDL R4, item_addr(SP) ; Step to next element.
      F1 53 F5 051A 1086 SOBGTR R3, NL1B ; Decrement and test count.
      04 051D 1087 RET ; Return to caller.
```



```
051E 1090      .SBTTL  ERR_HANDLER      - Exception handler for errors
051E 1091
051E 1092      :++
051E 1093      : ABSTRACT:
051E 1094      :
051E 1095      :     ERR_HANDLER accepts a signal and calls the ERR= and END=
051E 1096      :     error condition handler as if it were the CHF condition
051E 1097      :     facility itself. It passes along to FOR$$END_ERRHND
051E 1098      :     the ERR= and END= user addresses saved in the ISB at the
051E 1099      :     beginning of the I/O statement.
051E 1100
051E 1101      : FORMAL PARAMETERS:
051E 1102
051E 1103      :     NONE
051E 1104
051E 1105      : IMPLICIT INPUTS:
051E 1106
051E 1107      :     FOR$$A_CUR_LUB      Adr. of current logical unit block
051E 1108      :     ISB$$A_ERR_EQUAL    Adr. in user program to transfer to on errors or 0
051E 1109      :     ISB$$A_END_EQUAL    Adr. in user program to transfer to on EOF or 0
051E 1110
051E 1111      : IMPLICIT OUTPUTS:
051E 1112
051E 1113      :     NONE
051E 1114
051E 1115      : FUNCTION VALUE:
051E 1116
051E 1117      :     SS$_RESIGNAL to cause a resignal to occur (no END= or ERR=)
051E 1118      :     to give user handler and OTS default handler a chance at error.
051E 1119      :     However, if an ERR= or END= transfer is to be done, the function value is ig
051E 1120      :     by the condition handling facility because UNWIND has been called.
051E 1121
051E 1122      : SIDE EFFECTS:
051E 1123
051E 1124      :     If an ERR= or an END= transfer is to take place back to the user,
051E 1125      :     SYSS$UNWIND has been called to casue the condition handling facility
051E 1126      :     to unwind the stack when this error handler returns.
051E 1127      :--
051E 1128
051E 1129
051E 1130      ERR_HANDLER:
051E 1131      .WORD 0
051E 1132      MOVL  G^FOR$$A_CUR_LUB, R0
051E 1133      PUSHL #FOR$K_UNWINDPOP
051E 1134
051E 1135      PUSHL #0
051E 1136      PUSHL SP
051E 1137
051E 1138      PUSHAL ISB$$A_END_EQUAL(R0)
051E 1139      PUSHAL ISB$$A_ERR_EQUAL(R0)
051E 1140      MOVAL  16(SPT, -TSP)
051E 1141
051E 1142      PUSHL #4
051E 1143      PUSHL SP
051E 1144      MOVQ   sig_args(AP), -(SP)
051E 1145
051E 1146      CALLS  #3, L^FOR$$ERR_ENDHND
051E 1147
```

50 00000000'GF 0000 0520 1132 .WORD 0 ; no registers need saving
00 DD 0527 1133 MOVL G^FOR\$\$A_CUR_LUB, R0 ; R0 -> Current Control Block
00 DD 0529 1134 PUSHL #FOR\$K_UNWINDPOP ; make a long containing FOR\$K_UNWINDPOP
00 DD 0529 1135 PUSHL #0 ; to indicate UNWIND action is to pop LUB/IS
5E DD 052B 1136 PUSHL SP ; make a 0 by reference
FF78 C0 DF 052D 1137 ; point to the 0 - incremental depth =
FF74 C0 DF 052D 1138 ; no. of frames between user and establiher
7E 10 AE DE 0531 1139 PUSHAL ISB\$\$A_END_EQUAL(R0) ; push END= address
04 DD 0531 1139 PUSHAL ISB\$\$A_ERR_EQUAL(R0) ; push ERR= address
5E DD 0535 1140 MOVAL 16(SPT, -TSP) ; Indicate UNWIND action is to
7E 04 AC 7D 0539 1141 ; pop current LUB/ISB/RAB on error
00000000'EF 03 FB 0539 1142 PUSHL #4 ; 4 ENABLE args
00000000'EF 03 FB 053B 1143 PUSHL SP ; push address of ENABLE args
00000000'EF 03 FB 053D 1144 MOVQ sig_args(AP), -(SP) ; copy down the signal arg and
00000000'EF 03 FB 0541 1145 ; mechanism arg ptrs from the caller
00000000'EF 03 FB 0541 1146 CALLS #3, L^FOR\$\$ERR_ENDHND ; call the real handler

FOR\$IO_ELEM
2-047

```

; FORTRAN I/O element transmission M 7
ERR_HANDLER - Exception handler for erro 15-SEP-1984 23:53:43 VAX/VMS Macro V04-00 Page 27
[FORRTL.SRC]FORIOELEM.MAR;1 (20)

04 0548 1147 RET
    0549 1148
    0549 1149 .END
; end of module FOR$IO_ELEM.MAR

```


FOR\$IO_ELEM
Symbol table

; FORTRAN I/O element transmission N 7

15-SEP-1984 23:53:43 VAX/VMS Macro V04-00
6-SEP-1984 10:56:44 [FORRTL.SRC]FORIOELEM.MAR;1

Page 28
(20)

ACOM	00000487	R	02
ARBYTE	0000046D	R	02
ARLONG	0000047B	R	02
AROCTA	00000466	R	02
ARQUAD	00000482	R	02
ARRAY_CPLX_COM	000002ED	R	02
ARRAY_DC	000002E2	R	02
ARRAY_DC_GC_COM	000002EA	R	02
ARRAY_FC	000002DA	R	02
ARRAY_GC	000002E7	R	02
ARRAY_RET	0000031D	R	02
ARWORD	00000474	R	02
AWBYTE	00000435	R	02
AWLONG	00000443	R	02
AWOCTA	0000042E	R	02
AWQUAD	0000044A	R	02
AWWORD	0000043C	R	02
BNL1AY	0000050B	R	02
CALL1	000000D2	R	02
CALLUDF	000000CB	R	02
COM	000000C6	R	02
COM_IO_ELEM	00000034	R	02
COUNT	= 00000008		
CPX_FLAG	= 00000010		
DSC\$A_POINTER	= 00000004		
DSC\$B_DTYPE	= 00000002		
DSC\$K_CLASS_A	= 00000004		
DSC\$K_CLASS_S	= 00000001		
DSC\$K_DTYPE_B	= 00000006		
DSC\$K_DTYPE_BU	= 00000002		
DSC\$K_DTYPE_D	= 0000000B		
DSC\$K_DTYPE_DC	= 0000000D		
DSC\$K_DTYPE_F	= 0000000A		
DSC\$K_DTYPE_FC	= 0000000C		
DSC\$K_DTYPE_G	= 0000001B		
DSC\$K_DTYPE_GC	= 0000001D		
DSC\$K_DTYPE_H	= 0000001C		
DSC\$K_DTYPE_L	= 00000008		
DSC\$K_DTYPE_LU	= 00000004		
DSC\$K_DTYPE_T	= 0000000E		
DSC\$K_DTYPE_W	= 00000007		
DSC\$K_DTYPE_WU	= 00000003		
DSC\$L_ARSIZE	= 0000000C		
DSC\$W_LENGTH	= 00000000		
ELEM_ADR	= 00000004		
ELEM_VAL	= 00000004		
ERR_HANDLER	0000051E	R	02
FOR\$AA_UDF_PR1	*****	X	00
FOR\$AA_CUR_CUB	*****	X	00
FOR\$ERR_ENDHND	*****	X	00
FOR\$IO_B_R	00000007	RG	02
FOR\$IO_B_V	00000000	RG	02
FOR\$IO_DC_R	0000017D	RG	02
FOR\$IO_DC_V	00000175	RG	02
FOR\$IO_D_R	0000012D	RG	02
FOR\$IO_D_V	00000126	RG	02
FOR\$IO_FC_R	00000167	RG	02

FOR\$IO_FC_V	0000015F	RG	02
FOR\$IO_F_R	0000011A	RG	02
FOR\$IO_F_V	00000113	RG	02
FOR\$IO_GC_R	00000190	RG	02
FOR\$IO_GC_V	00000188	RG	02
FOR\$IO_G_R	00000140	RG	02
FOR\$IO_G_V	00000139	RG	02
FOR\$IO_H_R	00000153	RG	02
FOR\$IO_H_V	0000014C	RG	02
FOR\$IO_LO_R	00000107	RG	02
FOR\$IO_LU_V	00000100	RG	02
FOR\$IO_L_R	0000002B	RG	02
FOR\$IO_L_V	00000024	RG	02
FOR\$IO_T_DS	0000020A	RG	02
FOR\$IO_T_V_DS	0000021A	RG	02
FOR\$IO_WO_R	000000F4	RG	02
FOR\$IO_WU_V	000000ED	RG	02
FOR\$IO_W_R	00000019	RG	02
FOR\$IO_W_V	00000012	RG	02
FOR\$IO_X_DA	0000025B	RG	02
FOR\$IO_X_NL	0000035A	RG	02
FOR\$IO_X_SB	0000031E	RG	02
FOR\$IO_X_SE	000001ED	RG	02
FOR\$K_CLASS_SB	= 000000BF		
FOR\$K_UNWINDPOP	= 00000000		
IO_B_COM	0000000C	R	02
IO_CPLX_COM	0000019C	R	02
IO_DC_COM	00000183	R	02
IO_DC_GC_COM	00000199	R	02
IO_D_COM	00000132	R	02
IO_FC_COM	0000016D	R	02
IO_F_COM	0000011F	R	02
IO_GC_COM	00000196	R	02
IO_G_COM	00000145	R	02
IO_H_COM	00000158	R	02
IO_LO_COM	0000010C	R	02
IO_L_COM	00000030	R	02
IO_WO_COM	000000F9	R	02
IO_W_COM	0000001E	R	02
ISB\$A_END_EQUAL	= FFFFFFFF78		
ISB\$A_ERR_EQUAL	= FFFFFFFF74		
ISB\$B_STTM_TYPE	= FFFFFFFF71		
ISB\$K_FORSTTYLO	= 00000001		
ISB\$V_SNGLELEM	= 0000000C		
ISB\$W_STTM_STAT	= FFFFFFFF96		
ITEM_ADR	= 0000000C		
ITEM_SIZE	= 00000008		
ITEM_TYPE	= 00000004		
LUB\$A_BUF_END	= FFFFFFFFB4		
LUB\$A_BUF_PTR	= FFFFFFFFB0		
LUB\$V_UNFORMAT	= 00000009		
LUB\$W_UNIT_ATTR	= FFFFFFFFC		
NL1A	00000414	R	02
NL1AX	00000487	R	02
NL1AY	0000048B	R	02
NL1B	0000050E	R	02
NL1CAL	00000498	R	02

FOR\$IO_ELEM
Symbol table

; FORTRAN I/O element transmission B 8

15-SEP-1984 23:53:43 VAX/VMS Macro V04-00 Page 29
6-SEP-1984 10:56:44 [FORRTL.SRC]FORIOELEM.MAR;1 (20)

NL1PER	0000040B	R	02
NL2PER	000003C5	R	02
NLFMT	000003AD	R	02
NLUNIT	0000039F	R	02
RADDR	00000451	R	02
RBYTE	000000B0	R	02
RLONG	000000BC	R	02
ROCTA	000000A4	R	02
RQUAD	000000C2	R	02
RU	0000008F	R	02
RUBYTE	000004DD	R	02
RULONG	000004E9	R	02
RUOCTA	000004F5	R	02
RUQUAD	000004EF	R	02
RUWORD	000004E3	R	02
RWORD	000000B6	R	02
SF\$L_SAVE_PC	= 00000010		
SIG_ARGS	= 00000004		
STRIDE	= 0000000C		
T_DS_MASK	= 00000800		
UCOM	000004FF	R	02
WBYTE	00000075	R	02
WLONG	00000081	R	02
WOCTA	00000069	R	02
WQUAD	00000087	R	02
WUBYTE	000004B9	R	02
WULONG	000004C5	R	02
WUOCTA	000004D1	R	02
WUQUAD	000004CB	R	02
WUWORD	000004BF	R	02
WWORD	0000007B	R	02
XCALL1	0000008D	R	02
X_DA_MASK	= 0000081C		
X_SB_MASK	= 00000800		

+-----+
! Psect synopsis !
+-----+

PSECT name	Allocation	PSECT No.	Attributes														
. ABS .	00000000 (0.)	00 (0.)	NOPI	USR	CON	ABS	LCL	NOSHR	NOEXE	NORD	NOWRT	NOVEC	BYTE				
\$AB\$\$	00000000 (0.)	01 (1.)	NOPI	USR	CON	ABS	LCL	NOSHR	EXE	RD	WRT	NOVEC	BYTE				
_FOR\$CODE	00000549 (1353.)	02 (2.)	PIC	USR	CON	REL	LCL	SHR	EXE	RD	NOWRT	NOVEC	LONG				

+-----+
! Performance indicators !
+-----+

Phase	Page faults	CPU Time	Elapsed Time
Initialization	30	00:00:00.12	00:00:00.98
Command processing	120	00:00:00.64	00:00:04.77
Pass 1	236	00:00:05.94	00:00:21.40
Symbol table sort	0	00:00:00.82	00:00:01.73
Pass 2	248	00:00:02.53	00:00:08.61
Symbol table output	18	00:00:00.13	00:00:00.63

FOR\$IO_ELEM
VAX-11 Macro Run Statistics

; FORTRAN I/O element transmission C 8

15-SEP-1984 23:53:43 VAX/VMS Macro V04-00 Page 30
6-SEP-1984 10:56:44 [FORRTL.SRC]FORIOELEM.MAR;1 (20)

Psect synopsis output	2	00:00:00.03	00:00:00.12
Cross-reference output	0	00:00:00.00	00:00:00.00
Assembler run totals	656	00:00:10.22	00:00:38.29

The working set limit was 1500 pages.
35693 bytes (70 pages) of virtual memory were used to buffer the intermediate code.
There were 40 pages of symbol table space allocated to hold 591 non-local and 17 local symbols.
1149 source lines were read in Pass 1, producing 101 object records in Pass 2.
25 pages of virtual memory were used to define 12 macros.

+-----+
! Macro library statistics !
+-----+

Macro library name	Macros defined
-----	-----
\$255\$DUA28:[FORRTL.OBJ]FORRTL.MLB;1	3
\$255\$DUA28:[SYSLIB]STARLET.MLB;2	6
TOTALS (all libraries)	9

548 GETS were required to define 9 macros.

There were no errors, warnings or information messages.

MACRO/ENABLE=SUPPRESSION/DISABLE=(GLOBAL,TRACEBACK)/LIS=LIS\$:FORIOELEM/OBJ=OBJ\$:FORIOELEM MSRC\$:FORIOELEM/UPDATE=(ENH\$:FORIOELEM)+LI

0181 AH-BT13A-SE
VAX/VMS V4.0

DIGITAL EQUIPMENT CORPORATION
CONFIDENTIAL AND PROPRIETARY

FORINTUND
LIS

FORIOBEG
LIS

FORIOEND
LIS

FORLEX
LIS

FORMSG
LIS

FORMLTAB
LIS

FORINQUIR
LIS

FORIOELEM
LIS

FORIODATE
LIS

FORLIB
LIS